A flexible approach for automatic process decentralization using dependency tables

Walid Fdhila, Ustun Yildiz, Claude Godart

A flexible approach for automatic process decentralization using dependency tables

Walid Fdhila[1], Ustun Yildiz[2] and Claude Godart[1]

[1]LORIA - INRIA Nancy - Grand Est
F-54500 Vandœuvre-lès-Nancy, France
{*fdhilawa, godart*}*@loria.fr*

[2]University of California
Davis, CA 95616
yildiz@cs.ucdavis.edu

Abstract

Web service paradigm and related technologies have provided favorable means for the realization of collaborative business processes. From both conceptual and implementation points of view, the business processes are based on a centralized management approach. Nevertheless, it is very well known that the enterprise-wide process management where processes may span multiple organizational units requires particular considerations on scalability, heterogeneity, availability and privacy issues, that in turn, require particular consideration on decentralization. In this paper, our aim is to reconcile the decentralization of processes as a step towards the enterprise-wide solutions. We propose a methodology for transforming a centralized process specification into a form that is amenable to a distributed execution and to incorporate the necessary synchronization between different processing entities. The proposed technique has the advantage of being flexible that it computes the abstract constructs and provides a generalized approach to the decentralization of processes.

1 Introduction

With the emergence of the open standards, Web-based applications have become an intuitive support for Business-to-Business (B2B) and Business-to-Costumer (B2C) processes[24]. Particularly, Service Oriented Architectures (SOA) have the potential to enhance these by allowing autonomous and distributed business processes to interact with each other. Despite the decentralized nature of the context of B2B and B2C interactions, the conception and implementation of a typical business process rely on a centralized execution setting[1]. The relevant research literature on business management confirms that the decentralization is a critical need for several reasons[17][14][8][25][6]:

- scalability which is one of the pressing needs since many concurrent processes or instances of the same process are executed simultaneously, and a centralized architecture can cause a performance bottleneck,
- mutually equitable business relationships where no organization holds the control of the overall process,
- fault tolerance where different parts of a process can be executed even if some components fail,
- decentralization, since the systems are inherently distributed, not lend themselves to centralized control,
- distributing the data to reduce network traffic and improve transfer time as well as concurrence.

Although the decentralization fulfills most of the critical needs of collaborative business processes, it stops short in answering to the basic primitives of an execution setting such as the message routing between business partners to respect control dependencies. In the context of process decentralization, the existing works have differences and similarities in many different and strange ways, and modeling these accurately is difficult[17][14][8]. The developed techniques are often good for dealing with a particular aspect of decentralization rather than providing a generic and flexible manipulation setting required for process decentralization. The common limitation of the current decentralization approaches is their dependencies on the underlying process specification. They can deal with how the decentralized processes must be synchronized with the relevant messages of the low-level specification but they cannot address the fundamental questions about the decentralization of the combined control and data dependencies. Consequently, this becomes a major limitation for the use of these systems in different cases which are not explicitly specified in their decentralization mechanism.

In order to deal with the shortcomings of centralized process executions, we propose in this paper, a process decentralization technique that creates multiple sub-processes that interact with each other and thus, implements the dependencies of the centralized specification with P2P interactions. The main advantage of the developed approach is

the flexibility that it provides in terms of concepts and structures that it manipulates. This, in turn, allows the extension of the algorithms to the different needs of decentralization. In sharp contrast to previous works, our operation of decentralization computes the abstract process constructs, *i.e.*, workflow patterns[22]. This methodology separates the implementation details from the high-level reasoning that provides a more complete and generic solution to the decentralization problem. The technique uses a dependency table that resumes direct dependencies between the process activities. Next, it generates automatically transitive dependency tables resuming the transitive dependencies between activities having some common properties, *i.e.*, activities invoking the same service. It generates the corresponding sub-processes by specifying their mutual interconnections. Furthermore, we demonstrate the translation of the connectivity and communication between activities of the initial process to those between activities belonging to different sub-processes. We show also the reduction of the communication costs. The approach preserves also the semantics of the initial process specification.

The remainder of this paper is structured as follows. Section 2 describes the background and the issues involved in process decentralization. In section 3, we detail our approach and explain the different steps for getting the partitioned sub-processes. Approach flexibility is discussed in section 4. Finally, section 5 summarizes the ideas explained in the paper and outlines some future directions.

2 Related Work

In recent years, several approaches and architectures for decentralized process execution have been proposed. In the context of process partitioning, [18][8] are the first works that take on the challenge of partitioning BPEL processes. These contributions use program partitioning techniques in order to reduce the communication costs between derived process fragments. [14][13] present a similar approach to the decentralization focusing on the P2P interactions. Their contribution take on the formalization of the decentralized interactions from a conceptual point of view. Nevertheless, they present some specific BPEL examples rather than an overall approach that can decentralize any sophisticated process. [19] presents a similar approach to the decentralization of the control flow without considering data dependencies of process activities. Similar process partitioning approaches have been applied to different needs such as the implementation of secure interactions[3] or the decentralized exception handling[7]. In [27][26], the decentralization of processes has been studied in an abstract manner by extending the dead path elimination operation of workflow management systems. The decentralization focuses on the preservation of the centralized specification by preventing possible blocking situations.

Another approach to the decentralization concerns the implementation of additional applications to support the required interactions without embedding them into decentralized processes. ObjectFlow [10] uses a graph-based workflow definition model. Steps are executed by agents coordinated by a (potentially) distributed workflow engine which however accesses a centralized DBMS to store workflow states. In METEOR2 [20], process scheduling is distributed among various task managers. In Mentor [25], workflows are modeled using state-charts which are partitioned to each involved *processing entitiy* (PE). Each PE-specific statechart is executed locally on the PE workstation. Another example that support the decentralized execution without partitioning centralized specifications is Self-Serv[6]. In Self-Serv, the interactions of composed services are implicitly encoded within the processes.

In the context of Web services, [12] introduce the Web Services Choreography Description Language which has not received much attention, similarly to [2]. The organization for the Advancement of Structured Information Standards puts forward the ebXML standard for business collaboration [11]. More recently, service interaction patterns have been introduced in [4], and the language *Let's Dance* was introduced in [28]. The relationship between a global public process choreography and the private orchestrations is investigated in [23] based on work on process inheritance as introduced in [5]. The equivalence of process models, using their observable behavior, is studied in [21]. The relationship between compatibility notions in process choreography and consistency of process implementations with regards to behavioral interfaces is studied in [9].

3 Process Decentralization

In this section, we present our methodology for decentralizing a process specification characterizing a web service composition. Generally, a process is specified in an abstract way (e.g, using a graph based formalism) and mapped onto the architecture model level. We don't presume any particular process modeling approach, but simply assume that the basic elements of a process can be specified in an abstract way to be translated to an executable process language (i.e., process structure in terms of atomic activities and sub-processes, dependencies between the steps of a process, etc). Throughout this paper, we use the graph based formalism just for clarification reasons to guide the reader through the decentralization steps[16]. By definition, a process which specifies a web service composition defines the relationship between service invocations. This relationship may characterize either the control or data flow structure. Our approach takes into consideration both control and data dependencies between the process activities. Before explaining the steps to achieve decentralization, we

consider some assumptions and introduce some preliminary definitions.

- **Assumption**: Processes to be decentralized are structured [15]. This means that different activities are structured through control elements such as AND-split, OR-split, AND-join, OR-join..., and for each split element, there is a corresponding join element of the same type. Additionally, the split-join pairs are properly nested.

Definition 1 (Process) *A process, P, is a tuple ($\mathcal{A}$, $\mathcal{D}$, $\mathcal{E}_c$, $\mathcal{E}_d$, $\mathcal{S}_P$) where $\mathcal{A}$ is a set of activities, $\mathcal{D}$ is the set of data, $\mathcal{E}_c$ is a set of control edges with $\mathcal{E}_c \subset \mathcal{A}\times\mathcal{A}\times\mathcal{C}onds(\mathcal{D})$, $\mathcal{E}_d$ is a set of data edges with $\mathcal{E}_d \subset \mathcal{A}\times\mathcal{A}\times\mathcal{D}$ and $\mathcal{S}_P$ is a set of web services invoked by the the process activities.*

A process activity $a \in \mathcal{A}$ consists of a one-way or a bidirectional interaction with a service via the invocation of one of its operations. In conversational compositions, different operations of a service can be invoked with the execution of different activities. The set of activities that refer to the same service s_i is denoted $\mathcal{A}_{s_i}$. A control edge characterizes the mapping relationship while a data edge characterizes the mapping relation of the output and the input values of two activities. $\mathcal{S}_P$ is the set of services of a process $\mathcal{P}$.

Definition 2 (Activity) *an activity $a_i \in \mathcal{P}$ is a tuple ($\mathcal{I}n$, $\mathcal{O}ut$, s) where $\mathcal{I}n \subset \mathcal{D}$ is the set of a_i's inputs, $\mathcal{O}ut \subset \mathcal{D}$ is the set of a_i's outputs, $s \in \mathcal{S}$ is the invoked service by a_i*

Activities are related to each other and are dependent on each other. These dependencies are intra-process. Dependencies may also exist across processes and are referred to as inter-process dependencies.

Definition 3 (Preset) *The preset of an activity a_i, denoted $\bullet a_i$, is the set of activities which can be executed just before a_i and related to it by a control or data dependency.* $\bullet a_i = \{a_j \in \mathcal{A} | \exists e_{ji} \in \mathcal{E}_c \cup \mathcal{E}_d$ *s.t.* $source(e_{ji}) = a_j \wedge target(e_{ji}) = a_i\}$

Definition 4 (Postset) *The postset of an activity a_i, denoted $a_i \bullet$, is the set of activities which can be executed just after a_i and related to it by a control or data dependency.* $a_i \bullet = \{a_j \in \mathcal{A} | \exists e_{ij} \in \mathcal{E}_c \cup \mathcal{E}_d$ *s.t.* $source(e_{ij}) = a_i \wedge target(e_{ij}) = a_j\}$

A process is represented by a directed acyclic graph where nodes are activities and edges are data or control dependencies. Activities are depicted with boxes with the activity name inside and the web service it refers to. The arcs between boxes describe the dependencies. This leads to a well defined chronological execution of the activities. Next, we give an overview of the basic mechanisms of our method for the partitioning of process specifications.

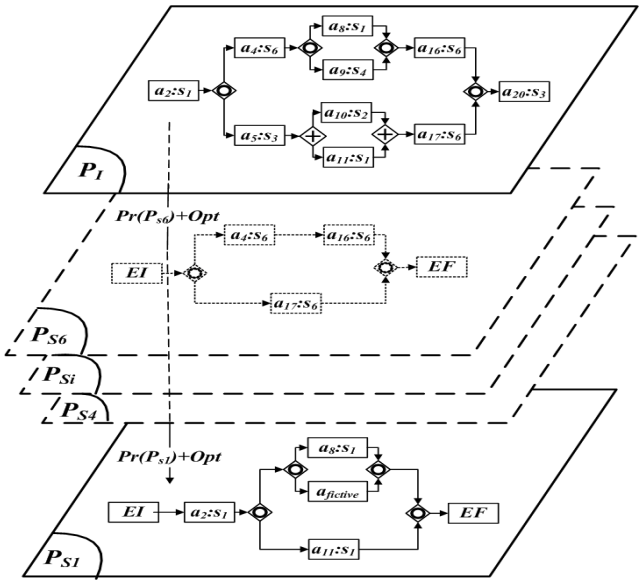

Figure 1. Process projection

3.1 Overview of the decentralization approach

The partitioning transforms the centralized process into behaviorally equivalent distributed sub-processes each of which related to a service. These partitions are executed independently at distributed locations (preferably collocated with the web services) and can be invoked remotely. They directly interact with each other using asynchronous messaging without any centralized control. The approach consists of four steps each explained in detail in the next sections, and guided by an illustrative example (cf. Fig. 2). The following is structured according to these steps, and for each step concepts and notations are introduced when required. The combinations of different dependencies between activities are analyzed, and the composite web service specification is partitioned using our program analysis technique. The number of final partitions depends on the number of invoked web services in the process. In other words, the decentralization can be considered as a projection of the initial process P_i on other plans P_{Si} each related to a service (cf. Fig. 1). Hence, each plan will contain a sub-process composed of activities invoking the same service as well as the transitive dependencies between them. Additional activities are added to maintain the semantics of the initial process and optimization techniques are applied. For instance, the sub-process related to s_1 is depicted by the projection of the initial process P_I on the plan P_{S_1}. To get a better understanding of the importance of decentralization in practice, let us consider the following process example depicted in figure 2. The graphical description includes control and data dependencies of process activities. Activities have identities such as a_i:s_j where a_i denotes the activity

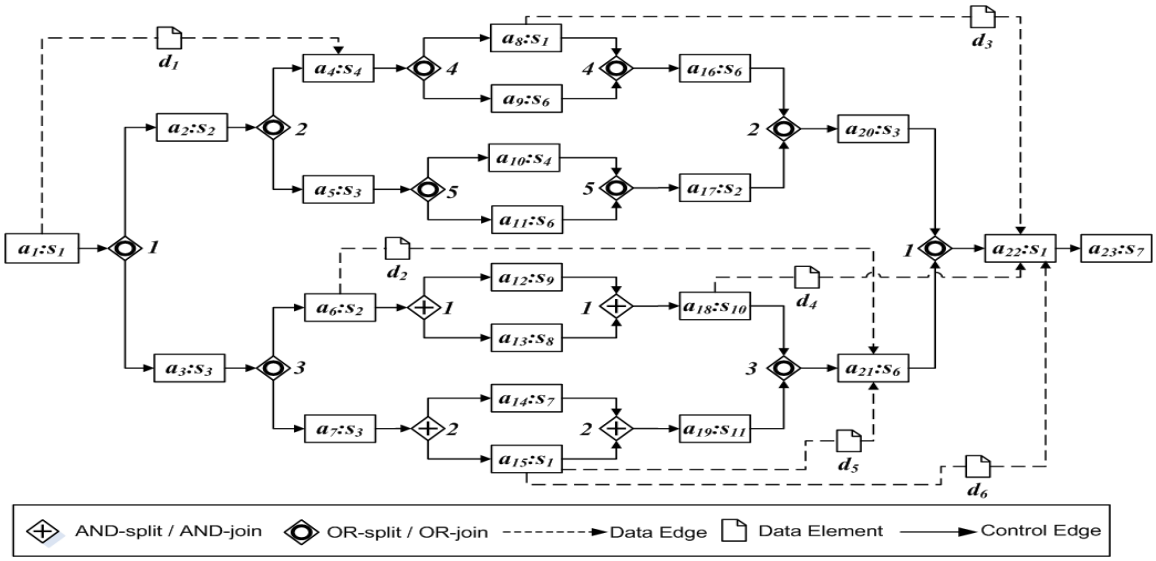

Figure 2. Motivating example

Table 1. Direct Control Dependency Table DCDT

	EI	$\mathbf{a_1 : s_1}$	$\mathbf{a_2 : s_2}$	$\mathbf{a_3 : s_3}$	$\mathbf{a_6 : s_2}$	$\mathbf{a_{12} : s_9}$	**...**	$\mathbf{a_{18} : s_{10}}$	$\mathbf{a_{21} : s_6}$	$\mathbf{a_{22} : s_1}$	**EF**
EI	-	*seq*	-	-	-	-		-	-	-	-
$\mathbf{a_1}$	-	-	OR_{1sp}	OR_{1sp}	-	-		-	-	-	-
$\mathbf{a_2}$	-	-	-	-	-	-		-	-	-	-
$\mathbf{a_3}$	-	-	-	-	OR_{3sp}	-		-	-	-	-
$\mathbf{a_6}$	-	-	-	-		AND_{1sp}		-	-	-	-
$\mathbf{a_{12}}$	-	-	-	-	-	-		AND_{1j}	-	-	-
...											
$\mathbf{a_{18}}$	-	-	-	-	-	-		-	OR_{3j}	-	-
$\mathbf{a_{21}}$	-	-	-	-	-	-		-	-	OR_{1j}	-
$\mathbf{a_{22}}$	-	-	-	-	-	-		-	-	-	*seq*

name and s_j the invoked service. We consider that services are known in advance.

3.1.1 Building Direct Dependency Tables (DCDT and DDDT)

The first step of our approach consists in building two tables $DCDT$ (Direct Control Dependency Table) and $DDDT$ (Direct Data Dependency Table) resuming all direct control and data dependencies between the activities. In other words, for each two activities which are control or data dependent, we note the patterns linking them. Each pattern has an identifier such as two corresponding join and split patterns with the same type have the same identifier (i.e. in fig. 2, $OR_{\mathbf{2sp}}$ and $OR_{\mathbf{2j}}$[1] have the same identifier 2). In the table, lines and columns are process activities $a_i \in \mathcal{P}$. The intersection between two activities corresponds to the control or data patterns between them if they exist. Table 1 resumes the control dependencies between process activities introduced in figure 2. We notice two extra activities EI and EF representing respectively the process initiating and ending activities. Their usability is that when partitioning the process, some sub-processes may have more than one activity in parallel initiating or ending them. In order to obtain structured sub-processes, and for a synchronization purpose, we add those two activities.

For example to build the table $DCDT$, we parse the process from EI until EF, and for each activity in the path we identify the control pattern linking it to each of its postset elements. If the pattern is a split-element we give it a new label. Otherwise (join-element), we look for the corresponding split-element and give it the same label. It should be noted that many patterns may link two control dependent activities. Table 1 is interpreted as follows:

- $line_i$: represents a_k, its postset and the control patterns

[1] $OR_j = OR_{join}$ and $OR_{sp} = OR_{split}$

linking them. For example, in $line_2$ the execution of a_1 which invokes the service S_1 implies the execution of both or one of the activities a_2 and a_3 which invoke respectively S_2 and S_3.

- $column_j$: represents a_k, its preset and the control patterns linking them. For example, in $column_5$ the execution of a_6 may or not begin after the termination of a_3 since they are related by an OR-split.
- $(line_i, column_j)$: corresponds to the control patterns, if they exist, that link activity a_k to a_l.

3.1.2 Building Transitive Control Dependency Tables (TCDT)

This step is an important phase in our approach as it considers the selection of the decentralization criteria (i.e. by services, by providers, by organizations...). In the following, we begin by decentralizing processes based on services. This means that every service will have a relative sub-process containing activities invoking it. We call the sub-process related to a service S_i as $\mathcal{P}_{s_i}$. Obviously, each sub-process must preserve a part of the initial process semantics. To cope with this, we infer from the direct Control dependencies table DCDT, the transitive control dependencies between activities invoking the same service. Next, we build a sub-process containing those activities and linked with the transitive patterns. This means that two transitively dependent activities are linked with the set of patterns which mainly exist in the path between them in the initial process (i.e. in Fig. 2, the transitive dependency between a_1 and a_8 is the path OR_{1sp}-OR_{2sp}-OR_{4sp}).

In this phase, we build a Transitive Control Dependency Table for each sub-process $\mathcal{P}_{S_i}$. To build the TCDTs, we proceed as follows: For each service, we select the set $\mathcal{A}_{S_i}$ of all activities invoking it, and we add EI and EF activities. For example, for the service $S1$ we have $\mathcal{A}_{S_1}$=$\{EI, a_1, a_8, a_{15}, a_{22}, EF\}$. We build a table $TCDT_i$ containing these activities in both lines and columns. As we don't know the order of precedence of those activities, we begin by EI and look for its transitive dependent activities a_j such as $a_j \in \mathcal{A}_{S_i}$. To achieve this, we look in the DCDT for the postset of EI. If there is at least one activity a_k in $EI\bullet$ such as $a_k \notin \mathcal{A}_{S_i}$, then we look for $a_k\bullet$ and continue until we find an activity $a_l \in \mathcal{A}_{S_i}$. For example, the transitive dependent activities of a_1 for $\mathcal{P}_{S_1}$ are a_8, a_{15} and a_{22}. In the same time, we have to save all control patterns linking an activity a_i to each of its transitive dependent activities and notice them in the table TCDT (i.e. the control patterns that link a_8 to a_{22} are $OR4j$, $OR2j$, $OR1j$). It may exist several control pattern paths which link two transitively dependent activities (i.e. a_1 is related to a_{22} by four control paths). Algorithm 1 gives a formal overview of the way we build TCDTs. The output of this algorithm is a TCDT for each service or sub-process. An example for the TCDT of the service $S1$ is resumed in the table 2 which is interpreted as follows:

Algorithm 1: Transitive Control Dependency Tables Building

Require: - DCDT // Direct Control dependency Table
- $\mathcal{S}$ // set of services invoked by the process
- $\mathcal{A}_{S_j}$ // set of activities invoking service S_j

for *each service* $S_j \in \mathcal{S}$ **do**
  Create $TCDT_{S_j}$ // *card*($\mathcal{A}_{S_j}$)×*card*($\mathcal{A}_{S_j}$)
  $Current\_activity \leftarrow EI$
  $Current\_set \leftarrow \{EI\}$
  while $Current\_set \neq \{\emptyset\}$ **do**
    Delete $Current\_activity$ *from* $Current\_set$
    $Postset \leftarrow (Current\_activity)\bullet$
    $TDep\_set \leftarrow \{\emptyset\}$
    for *each* $a_i \in Postset$ **do**
      $Ctr_{a_i} \leftarrow DCDT(Current\_activity, a_i)$
      Add (a_i, Ctr_{a_i}) *to* $TDep\_set$
    repeat
      for *each* $(a_i, Ctr_{a_i}) \in TDep\_set$ **do**
        if $a_i \in \mathcal{A}_{S_j}$ **then**
          Add Ctr_{a_i} *to* $TCDT_{S_j}(Current\_activity, a_i)$
          if $a_i \notin Current\_set$ *and* $a_i \neq EF$ **then** *Add* a_i *to* $Current\_set$
        else
          for *each* $a_k \in (a_i)\bullet$ **do**
            $Ctr_{a_k} \leftarrow Ctr_{a_i} + DCDT(a_i, a_k)$
            if $(a_i, Ctr_{a_i}) \notin TDep\_set$ **then** *Add*(a_i, Ctr_{a_i}) *to* $TDep\_set$
        Delete (a_i, Ctr_{a_i}) *from* $TDep\_set$
    until $TDep\_set = \{\emptyset\}$
    $Current\_activity \leftarrow$ *First_elem*$(Current\_set)$

Result: TCDT for each service $S_j \in \mathcal{S}$

- $line_i$: represents the transitive postset of a_k and the set of control flow patterns linking them. The transitive postset of a_1 is $\{a_8, a_{15}, a_{22}\}$
- $column_j$: represents the transitive preset of a_k and the set of control flow patterns linking them. The transitive preset of a_{22} is $\{a_1, a_8, a_{15}\}$
- $(line_i, column_j)$: corresponds to the set of control flow patterns that links activity a_k to a_l if it exists.

3.1.3 Building sub-processes

In this section, we show how to build a sub-process for each $TCDT_{S_j}$ we obtained in the last step. Each sub-process represents the control flow between activities invoking the same service. Algorithm 2 explains formally how to proceed to achieve this. For a control connector ctr, $\overline{ctr}$ characterizes corresponding split or join connector of the same type. For example if ctr is an OR_j, $\overline{ctr}$ is an OR_{sp} that corresponds to ctr in the process model. First, we optimize

Table 2. Transitive Control Dependency Table for service S1 : $TCDT_{S1}$

	EI	$\mathbf{a_1}$	$\mathbf{a_8}$	$\mathbf{a_{15}}$	$\mathbf{a_{22}}$	**EF**
EI		*seq*	-	-	-	-
a1	-	-	$OR_{1sp}, OR_{2sp}, OR_{4sp}$	$OR_{1sp}, OR_{3sp}, \mathbf{AND_{2sp}}$	$OR_{1sp}, OR_{2sp}, OR_{4sp}, OR_{4j}, OR_{2j}, OR_{1j}$ $OR_{1sp}, OR_{2sp}, \mathbf{OR_{5sp}}, \mathbf{OR_{5j}}, OR_{2j}, OR_{1j}$ $OR_{1sp}, OR_{3sp}, \mathbf{AND_{1sp}}, \mathbf{AND_{1j}}, OR_{3j}, OR_{1j}$ $\mathbf{OR_{1sp}}, \mathbf{OR_{3sp}}, \mathbf{AND_{2sp}}, \mathbf{AND_{2j}}, \mathbf{OR_{3j}}, \mathbf{OR_{1j}}$	-
$\mathbf{a_8}$	-	-	-	-	$OR_{4j}, OR_{2j}, OR_{1j}$	-
$\mathbf{a_{15}}$	-	-	-	-	$\mathbf{AND_{2j}}, OR_{3j}, OR_{1j}$	-
$\mathbf{a_{22}}$	-	-	-	-	-	*seq*

each table $TCDT_{S_j}$, by deleting every two corresponding split and join patterns having no activities $a_i \in \mathcal{A}_{S_j}$ between them (i.e. in table 2 corresponding to $TCDT_{S_1}$ we delete OR_{5sp} and OR_{5j} since a_{10} and $a_{11} \notin \mathcal{A}_{S_1}$). We also delete, for a given $TCDT_{S_j}$, paths which contain two corresponding $AND_{K_{sp}}$ and AND_{K_j} which have an acitvity $a_i \in A_{S_j}$ between them (i.e. in $TCDT_{S_1}$ we delete the fourth path linking a_1 to a_{22} since between AND_{2sp} and AND_{2j} we cannot pass by a_{14} to a_{22} without executing a_{15} which $\in \mathcal{A}_{S_1}$).

Schema 3 explains the optimization process. The bold arrows characterize the transitive dependencies between activities invoking the service S_1. The two boxes represent the patterns to delete from the paths $T\_link_{1-22}(b)$ and $T\_link_{1-22}(c)$ since a_{10}, a_{11}, a_{12} and $a_{13} \notin \mathcal{A}_{S_1}$. Also the path $T\_link_{1-22}(d)$ has to be deleted since it passes by a_{19} which executes only if a_{15} is executed and $a_{15} \in \mathcal{A}_{S_1}$. This link is replaced by T_link_{1-15} and T_link_{15-22}. This optimization process avoids having unusable synchronization points.

Once optimization is achieved, for each $TCDT_{S_j}$ we build the corresponding sub-process by connecting its composing activities by the set of control flow patterns represented by $TCDT_{S_j}(a_i, a_k)$. The sub-process building algorithm is ascendent which means that we begin by EI until reaching EF. In each step we look for the transitive postset of the current activity and the correspondent control path linking it to each of this postset elements. To continue with the same example, the resulting sub-process P_{S1} is depicted in the figure 4. Activities named $a_{fictive}k$ are activities which execution time is zero, and used to synchronize and maintain the semantics of the process. To resume, the output of this step is a sub-process for each service and which will be executed by a separate engine.

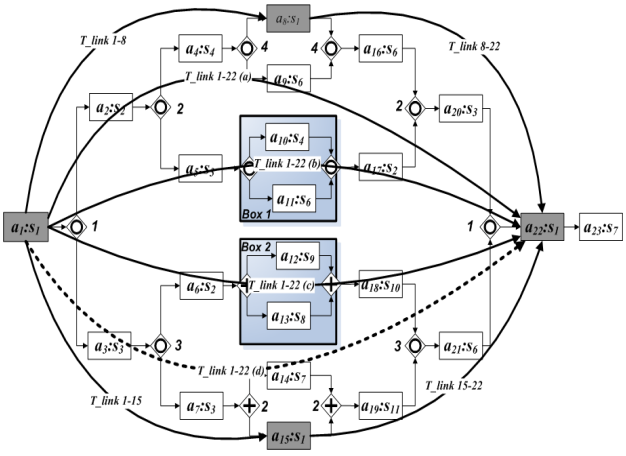

Figure 3. Optimization process for P_{S_1}

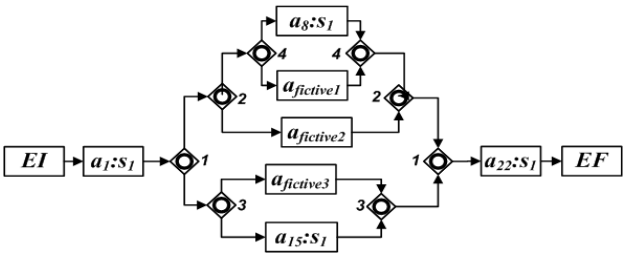

Figure 4. sub-process P_{S_1} **of service** $S1$

3.1.4 Interconnecting sub-processes

In this section, we present how to translate the connectivity and communication between activities of the initial process to that between activities belonging to different sub-processes. Both data and control dependencies are taken into consideration. For each activity a_i of the initial process, we look from the DCDT for the control dependencies with its postset. According to the patterns linking a_i with each of $a_i\bullet$ elements, we choose the right connection. In the last three sections, most patterns are taken into consideration. However, in the synchronization phase, we consider only the basic patterns such as AND_{sp}, AND_j, OR_{sp}, OR_j, XOR_{sp}, XOR_j and $Sequence$. Algorithm 3 gives a formal overview of the interconnection process. For a control connector ctr, $\overline{ctr}\bullet$ is the first activity following $\overline{ctr}$. While $a_{i_{ctr}}\bullet$ represents the postset of a_i through ctr (i.e. in Fig. 2, if a_4 doesn't exist ($a_{2_{\{OR_{2sp}OR_{4sp}\}}}$)$\bullet$ =$\{a_8, a_9\}$). First, we

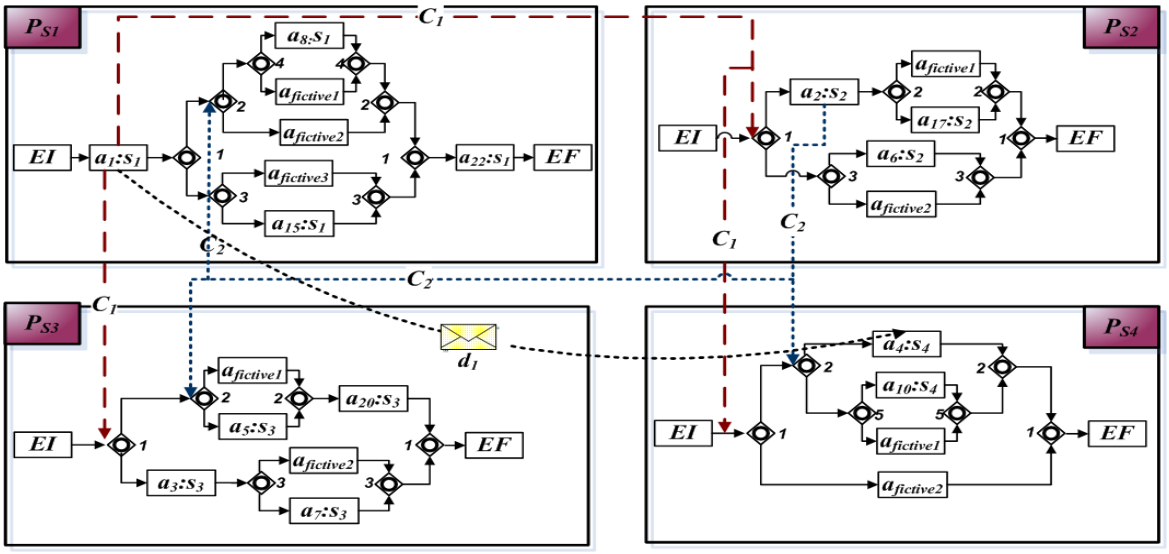

Figure 5. sub-processes interconnection

synchronize the control flow between activities belonging to different sub-processes. We distinguish two cases treated by the algorithm: (i) when two activities a_i and a_j are linked by only one control pattern, (ii) and when they are linked by a set of control patterns. Next, we consider data flow connections. Data dependent activities are activities which exchange data. For instance, in fig.2, a_4 needs the data d_1 from a_1 to be executed. As a_1 and a_4 belong to different sub-processes P_{S1} and P_{S4}, we have to connect them. The connection is a message exchange where a_1 is the sender and a_4 is the receiver (cf. Fig. 5).

Figure 5 illustrates a part of the interconnection of sub-processes P_{S_1}, P_{S_2}, P_{S_3} and P_{S_4} issued from the decentralization of the process introduced in figure 2. In each of the four sub-processes, we note the presence of OR_{1sp} which depends on the decision taken by a_1. Hence, the presence of three dashed arrows C_1 connecting a_1 to each of these patterns. The advantage of connecting a_1 to the patterns instead of $a_1\bullet$=$\{a_2$,$a_3\}$ is the minimization of the exchanged messages number. Typically, if we connect a_1 only to $a_1\bullet$ and consider that a_3 won't be executed during runtime. In this case, a_3 must inform a_6 and a_7 to be skipped, which may be in other sub-processes and in turn, have to inform their postsets and so on. This leads to a huge number of unusable synchronization messages. However, the other sub processes *knew* already that a_3 won't be executed as they had received the decision of a_1. For example, the reception of C_1 by P_{S_3} and P_{S_2} just before OR_{1sp} causes automatically the non-execution of a_3 and all its next activities until OR_{1j} in P_{S_3} and a_6 and its next activities in P_{S_2}. Therefore, a_3 doesn't need to inform $a_3\bullet$ of its non-execution, which in turn, doesn't inform $(a_3\bullet)\bullet$ and this reduces considerably *inter-sub-processes* message exchanges. If we consider that $a_{22}\notin P_{S_1}$, and not preceded by $\overline{OR_{1sp}}$, then we have to connect a_1 to a_{22}= $\overline{OR_{1sp}}\bullet$. Because a_{22} has to know the number of control messages it has to wait to be executed. Therefore, if a_3 is not executed then a_{22} knows in advance that it won't receive a message from a_{21} (cf. Fig. 2). The same scenario is done with $a_2\in P_{S_2}$ as it is followed by OR_{2sp} (connexion C_2 in fig 5).

If an activity a_i is connected to its postset by Seq, AND_{sp}, or a set of join-elements then we simply connect a_i to $a_i\bullet$. Thanks to our synchronization mechanism, each of $a_i\bullet$ knows in advance if it has to wait or not for a message from a_i. The last scenario is when a_i is connected to $a_i\bullet$ by a set of split-elements ctr. In this case, if there is no OR_{sp} in ctr then we simply connect a_i to $a_i\bullet$. Otherwise, we connect a_i to each $OR_{id_{sp}}\in ctr$ and we add a connection between a_i and $\overline{OR_{id_{sp}}}\bullet$ if it exists. Note that an $OR_{id_{sp}}$ may belong to many sub-processes in the same time.

During runtime, activities must respect all their preconditions and postconditions before and after execution. Preconditions are control and data connections with each of its preset activities whereas postconditions are control and data connections with each of its postset activities (i.e. in fig.5, preconditions of a_4 are C_1, C_2 and d_1). Dependent activities may be on other sites or sub-processes. As we mentioned earlier, a connection may be as a message exchange. The latter may be skipped during execution (i.e. in fig 2, if a_3 is not executed, there is no data transfer between a_6 and a_{21} since both of them won't be executed.) In some cases, a message must be sent even if the activity is not executed. For instance, even if a_3 is not executed, a_{15} must inform a_{22} to not wait for d_6 since a_{22} execution doesn't depend on a_3 execution. Each message must hold an instance identifier to make correlation between process instances.

Algorithm 2: sub-processes building

Require: -TCDT // Transitive dependency Tables
for *each* $TCDT_{S_j}$ *sas* $S_j \in \mathcal{S}$ **do**
  // $TCDT_{S_j}$ Optimisation
  for *each* $(a_i, a_j) \in TCDT_{S_j}$ **do**
    for *each* $path_k \in TCDT_{S_j}(a_i, a_j)$ **do**
      if $\exists$ *(ctr,* $\overline{ctr}$*)*$\in path_k$ *sas* $\nexists\, a_n \in \mathcal{A}_{S_j}$ *between ctr and* $\overline{ctr}$ **then**
        Delete (ctr, $\overline{ctr}$) from $path_k$
  for *each* $(a_i, a_j) \in TCDT_{S_j}$ **do**
    for *each* $path_k \in TCDT_{S_j}(a_i, a_j)$ **do**
      if $\exists$ *consecutive* AND_{id}, $\overline{AND}_{id} \in path_k$ **then**
        Delete $path_k$
  for *each* $a_i \in TDT_{S_j}$ **do**
    if $\exists AND_{id} \in$ *Line*(a_i) *sas card*(And_{id})*=1* **then**
      Delete AND_{id}
  // $TCDT_{S_j}$ related process building
  for *each* $(a_i, a_j) \in TDT_{S_j}$ **do**
    for *each* $path_k \in TDT_{S_j}(a_i, a_j)$ **do**
      connect *(*$(a_i, a_j, path_k)$*)*
      if $\exists$ *consecutive ctr,* $\overline{ctr}$ **then**
        Add fictive activity

Result: a sub-process for each $TCDT_{S_j}$

4 Approach flexibility to support different criteria

In the last three sections we presented techniques to partition a process according to services. This assumes that all business partners or services are assigned to activities before the process instantiation. However, in some cases, the business partners that take part in the instance of a given process may not be known in advance, but selected at runtime. Besides, corporations may operate across organizational boundaries. Consequently, collaborations involve several autonomous organizations which share the same public business process. Partitioning the latter, over the organizations involved, optimizes the flow of work and gives them more autonomy to create or modify the process at any time. To cope with these requirements, other decentralization criteria should be considered. This means that the decentralization is not done according to services, but according to organizations. This approach requires the generation of a sub-process for each organization rather than the generation of a sub-process for each service. This reduces the number of synchronization messages, since the number of sub-processes decreases. It also features the runtime assignment of business partners to process activities in order to provide the adequate flexibility to support dynamic collaborations of business partners. Furthermore, it can improve security conditions such as privacy, since each organization controls its proper process. As a result, the extension of the approach to the different organizational needs demonstrates its flexibility. For the above example, the same algorithms can be simply implemented by grouping activities belonging to the same organization. Hence, instead of building a TCDT for each service, we can build a TCDT for each organization and then the corresponding sub-process. This requires the modification of the same algorithms introduced in section 3. Within the algorithms, the modification of services S_j by another criterion (for instance organizations O_j or providers P_j...) is enough to switch from one decentralization criterion to another. The decentralization based on services or providers is more suitable when only one organization is involved in the business process. The decentralization according to providers may be more efficient than to services, if the set of services involved in the process that a provider supports, is located on the same site.

Algorithm 3: Sub-processes interconnection

Require: -DCDT // Direct Control Dependency Table
-DADT // Direct Data Dependency Table
-The set of sub-processes
for *each* $a_i \in \mathcal{A}$ **do**
  // Control dependencies interconnection
  $CTR_{a_i} \leftarrow \{\emptyset\}$
  for *each* $a_j \in (a_i)\bullet$ *in DCDT sas* $a_j \notin P_{s_{a_i}}$ **do**
    if $ctr_{a_{ij}} \notin CTR_{a_i}$ **then**
      $CTR_{a_i} \leftarrow CTR_{a_i} \bigcup ctr_{a_{ij}}$
  for *each* $ctr \in CTR_{a_i}$ **do**
    if $ctr \in \{Seq, AND_{sp}\} \bigcup$ *set of join-elements* **then**
      connect(a_i, $a_{i_{ctr}}\bullet$*)*
    if $ctr = OR_{sp_{id}}$ *or* $ctr = XOR_{sp_{id}}$ **then**
      for *each* P_{s_k} *sas* $\exists\, ctr_k = ctr$ **do**
        connect(a_i, ctr_k*)*
      connect(a_i, $\overline{ctr}\bullet$*)*
    if ctr *is a set of split-elements* **then**
      if $\nexists\, OR_{sp_{id}} \in ctr$ **then**
        connect(a_i, $a_{i_{ctr}}\bullet$*)*
      else
        for *each* $OR_{sp_{id}} \in ctr$ **do**
          for *each* P_{s_k} */* $\exists\, ctr_k = OR_{sp_{id}}$ **do**
            connect(a_i, ctr_k*)*
          connect(a_i, $\overline{OR_{sp_{id}}}\bullet$*)*
  // data dependencies interconnection
  Lookup in DADT for $a_i\bullet$
  connect (a_i, $a_i\bullet$*)*

Result: Interconnected sub-processes

5 Conclusion

This paper has presented an approach to the flexible decentralization of process specifications. The developed approach is applicable to a wide variety of service composition standards that follow the process management approach such as WS-BPEL. In contrast to previous works

that take on the process decentralization approaches, our approach is based on the computation of very basic dependencies between process elements that provides a considerable level of understanding and also the flexibility for further manipulation. The computation of basic dependencies have led, in turn, to the re-implementation of the semantics of a centralized specification with peer-to-peer interactions among the derived decentralized process specifications. In addition to the simplicity, the flexibility allows the efficient re-instanciation of the algorithms for the different needs of decentralization and also dynamic collaborations where services are discovered at runtime. We are planning to extend the current approach to support advanced patterns and fault handlers. Our future work includes implementation of the introduced methodology on a web service composition language and a quantitative evaluation of the approach in terms of message exchanges. Another challenge is to take into consideration security aspects between the decentralized process specifications.

References

[1] Workflow management coalition: process denition interchange v 1.0 nal. http://www.wfmc.org, 1998.

[2] S. F. Arkin, Sid Askary and W. Jekeli. Web service choreography interface (wsci) 1.0,, 2002.

[3] V. Atluri, S. A. Chun, R. Mukkamala, and P. Mazzoleni. A decentralized execution model for inter-organizational workflows. *Distributed and Parallel Databases*, 22(1):55–83, 2007.

[4] A. P. Barros, M. Dumas, and A. H. M. ter Hofstede. Service interaction patterns. In *Business Process Management*, pages 302–318, 2005.

[5] T. Basten and W. M. P. van der Aalst. Inheritance of behavior. *J. Log. Algebr. Program.*, 47(2):47–145, 2001.

[6] B. Benatallah, Q. Z. Sheng, and M. Dumas. The self-serv environment for web services composition. *IEEE Internet Computing*, 7(1):40–48, 2003.

[7] G. Chafle, S. Chandra, P. Kankar, and V. Mann. Handling faults in decentralized orchestration of composite web services. In *ICSOC*, pages 410–423, 2005.

[8] G. Chafle, S. Chandra, V. Mann, and M. G. Nanda. Decentralized orchestration of composite web services. In *WWW (Alternate Track Papers & Posters)*, pages 134–143, 2004.

[9] G. Decker and M. Weske. Behavioral consistency for b2b process integration. In *CAiSE*, pages 81–95, 2007.

[10] M. Hsu and C. Kleissner. Objectflow: Towards a process management infrastructure. *Distributed and Parallel Databases*, 4(2):169–194, 1996.

[11] S. S. A. Jean-Jacques Dubray and M. J. Martin. *ebXML Business Process Specification Schema Technical Specification v2.0.4. OASIS*. 2006.

[12] N. Kavantzas, D. Burdett, G. Ritzinger, and Y. Lafon. Web services choreography description language version 1.0. http://www.w3.org/TR/ws-cdl-10, 2004.

[13] R. Khalaf, O. Kopp, and F. Leymann. Maintaining data dependencies across bpel process fragments. *Int. J. Cooperative Inf. Syst.*, 17(3):259–282, 2008.

[14] R. Khalaf and F. Leymann. E role-based decomposition of business processes using bpel. In *ICWS*, pages 770–780, 2006.

[15] B. Kiepuszewski, A. H. M. ter Hofstede, and C. Bussler. On structured workflow modelling. In *CAiSE*, pages 431–445, 2000.

[16] F. Leymann and D. Roller. *Production Workflow - Concepts and Techniques*. PTR Prentice Hall, 2000.

[17] F. Montagut, R. Molva, and S. T. Golega. The pervasive workflow: A decentralized workflow system supporting long-running transactions. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 38(3):319–333, 2008.

[18] M. G. Nanda, S. Chandra, and V. Sarkar. Decentralizing execution of composite web services. In *OOPSLA*, pages 170–187, 2004.

[19] W. Sadiq, S. W. Sadiq, and K. Schulz. Model driven distribution of collaborative business processes. In *IEEE SCC*, pages 281–284, 2006.

[20] A. P. Sheth, K. Kochut, J. A. Miller, D. Worah, S. Das, C. Lin, D. Palaniswami, J. Lynch, and I. Shevchenko. Supporting state-wide immunisation tracking using multiparadigm workflow technology. In *VLDB*, pages 263–273, 1996.

[21] W. M. P. van der Aalst, A. K. A. de Medeiros, and A. J. M. M. Weijters. Process equivalence: Comparing two process models based on observed behavior. In *Business Process Management*, pages 129–144, 2006.

[22] W. M. P. Van Der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distrib. Parallel Databases*, 14(1):5–51, 2003.

[23] W. M. P. van der Aalst and M. Weske. The p2p approach to interorganizational workflows. In *CAiSE*, pages 140–156, 2001.

[24] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. F. Ferguson. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. PRS Prentice Hall, 2005.

[25] D. Wodtke, J. Weißenfels, G. Weikum, and A. K. Dittrich. The mentor project: Steps toward enterprise-wide workflow management. In *ICDE*, pages 556–565, 1996.

[26] U. Yildiz and C. Godart. Centralized versus decentralized conversation-based orchestrations. In *Proceedings of the 9th IEEE International Conference on E-Commerce Technology (CEC 2007) / 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services (EEE 2007), CEC/EEE*, pages 289–296, 2007.

[27] U. Yildiz and C. Godart. Towards decentralized service orchestrations. In *Proceedings of the 2007 ACM Symposium on Applied Computing , SAC*, pages 1662–1666, 2007.

[28] J. M. Zaha, A. P. Barros, M. Dumas, and A. H. M. ter Hofstede. Let's dance: A language for service behavior modeling. In *OTM Conferences (1)*, pages 145–162, 2006.